Käyttäjäkokemuksen parantaminen web-sovelluksissa AJAX:n avulla

Mikko Tikkanen
mikko.tikkanen@gmail.com

Tiivistelmä

Nykypäivänä web-sovellusten käyttöaste on nousemassa työpöytäsovellusten rinnalle; miltei jokainen tietokonetta käyttävä hyödyntää web-sovelluksia, usein jopa päivittäisessä elämässään. Perinteisesti suurin ongelma web-sovellusten hyväksyttävyydessä on ollut työpöytäsovelluksia huonompi käyttäjäkokemus: sivuja joudutaan lataamaan jatkuvasti uudelleen, monimutkaistaa käyttöliittymää ja tekee siitä hitaan käyttää. Tässä dokumentissa esitellään erilaisia tapoja Ajax-tekniikoiden hyödyntämiseksi käyttäjäkokemuksen parantamisessa.

Dokumentissa esiteltyjen tutkimustulosten lisäksi tein myös esimerkkiprojektin, joka demonstroi joitain tutkimuksessa esiin tulleita ratkaisuja.

Summary

Nowdays, web-applications are almost as common as desktop applications; almost every, computer using, person is using web-applications, often at everyday life. Traditionally, the biggest acceptabiltiy problem in web-applications has been, compared to desktop applications, the ”not as good” user experience; pages have to be constantly reloaded, which complicates the user interface and makes the application slow and cumbersome to use.This document proposes different ways to solve the user experience problem, using AJAX techniques.

In addition to the results shown in this document, there was an case study project, which demonstrates some of the solutions that appeared in the research.

Sisällysluettelo

1 Johdanto

Nykypäivänä web-sovellusten käyttöaste on nousemassa työpöytäsovellusten rinnalle; miltei jokainen tietokonetta käyttävä hyödyntää web-sovelluksia*, usein jopa päivittäisessä elämässään. Perinteisesti suurin ongelma web-sovellusten hyväksyttävyydessä on ollut työpöytäsovelluksia huonompi käyttäjäkokemus: sivuja joudutaan lataamaan jatkuvasti uudelleen, mikä puolestaan monimutkaistaa käyttäjäkokemusta ja tekee siitä yleisesti hitaan käyttää.

Alkuperäisenä tarkoituksenani opinnäytetyölleni oli toteuttaa projektina nykyaikainen web-palvelu sekä kirjallinen osuus palvelun suunnittelusta ja toteutuksesta, aiheeksi palvelulleni valitsin sosiaalisen uutispalvelun. Pian kuitenkin kävi selväksi, että kyseinen palvelu vaatii laajan käyttäjämäärän toimiakseen tehokkaasti, täytyi palvelun suunnittelussa ja toteutuksessa ottaa huomioon eräitä teknisiä seikkoja (HTTP kyselyiden ja suorituskyvyn optimointi, rakenteen skaalautuvuus jne.), jotta palvelu ei tukehtuisi käyttäjämäärien kasvaessa. Koska nämä asiat on otettava huomioon alusta lähtien, tai toteutettava koko palvelu myöhemmin uudelleen, päädyin tekemään kattavia profilointeja eri toteutustapojen välillä. Paria kuukautta, ja noin viisi sovellus iteraatiota myöhemmin, demoalusta (Cloud) oli valmis ja aloin kirjoittaa suunnitelmaa kirjalliselle osuudelleni. Ensimmäisissä tarkasteluissa, ohjaajan kanssa, tulimme kuitenkin tulokseen, että projektin kasvettua niinkin suureksi, tulisi koko projektista tehtävästä kirjallisesta raportista liian laaja-alainen, niinpä päädyimme siihen tulokseen, että tekisin projektiraporttini AJAX:n hyödyntämisestä web-palvelujen käyttettävyyden sekä käyttäjäkokemuksen parantamisessa.

Dokumentti myös esittelee projektissa luotuja, nykyaikaisia AJAX-tekniikoita hyödyntäviä ratkaisumalleja.

* Web sovellus on interenetin kautta jaeltava sovellus. Yleensä web-sovellus on jaettu osiin niin, että käyttöliittymä sijaitsee käyttäjän selaimessa, itse käyttölogiikan pyöriessä palvelimella. Web-sovellukseksi voidaan laskea esimerkiksi web-sähköposti.

2 AJAX, mikä se on?

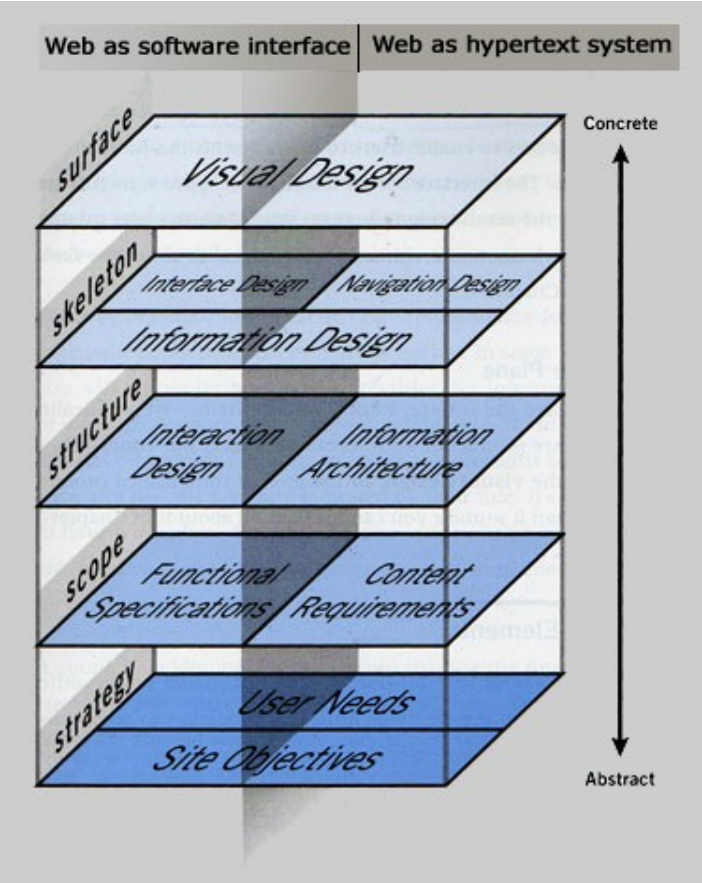
AJAX (lyhenne sanoista Asynchronous JavaScript And XML) tarkoitti alkuperäisessä merkityksessään tekniikkaa, jossa web-sovellus hakee, JavaScriptiä hyödyntäen, XML

muotoista dataa palvelimelta asynkronisesti, ilman sivun lataamista uudelleen. (Wikipedia 2009) Nykyään AJAX:lla viitataan yleisesti JavaScript pohjaisiin tekniikoihin, joilla web-sovelluksen taustalla vaihdetaan pieniä määriä, eri muotoista (XML, JSON, HTML jne), dataa palvelimen kanssa.

3 Käyttäjäkokemus

Jesse James Garretin määrittelee kirjassaan käyttäjäkokemuksen seuraavasti, ”*Kuinka tuote käyttäytyy ja sitä käytetään oikeassa maailmassa*”. (Garrett, Jesse James 2003) (s. 10) Yksinkertaistettuna voidaan siis puhua sateenvarjotermistä, joka kattaa käyttäjän kokonaisvaltaisen kokemuksen tuotteesta ja sen käyttämisestä.

Jesse James Garrett on myös luonut ehkä tunnetuimman kaavion käyttäjäkokemuksen rakenteesta web-sivustoilla/-sovelluksissa. Kaaviossa käyttäjäkokemus on jaettu viiteen eri tasoon: *Strategia, laajuus, rakenne, luuranko ja pinta* (Strategy, scope, structure, skeleton, surface). Kaavio siis helpottaa käyttäjäkokemuksen suunnittelua, jakaen sen erillisiin tasoihin, joista ylempi taso pohjautuu aina alempaan tasoon. Näin esimerkiksi käyttäjäkokemuksen suunnittelu voidaan aloittaa selkeistä abstraktioista (kuten esimerkiksi käyttäjien tarpeet), edeten kohti konkreettisempia toteutuksia (esimerkiksi vuorovaikutteisuuden suunnittelu). (Garrett, Jesse James 2003) (s. 21-36)



Kuva 1: Käyttäjäkokemuksen viisi tasoa (Garrett, Jesse James 2003)

Tässä työssä keskitytään kyseessä olevan kaavion neljänteen (Skeleton) tasoon, sillä AJAX:n hyödyt saavutetaan pääasiallisesti tällä tasolla – Käyttöliittymäsuunnittelu (Interface design) -laatikossa. Lisäksi AJAX tarjoaa teknisen edun, jolla käyttöliittymän vasteaikoja saadaan parannettua.

3.1 Käyttöliittymä

Klassisen rakenteen mukaisesti, jonkin sivun osa-alueen muuttuessa, koko sivu täytyy ladata uudelleen. Tästä johtuen esimerkiksi interaktiiviset lomakkeet ovat olleet miltei mahdottomia toteuttaa, ja käyttäjä on yleensä pakotettu lähettämään lomake palvelimelle, jossa PHP-sivu tarkastaa lomakkeen tiedot ja antaa käyttäjälle palautteen. Poikkeuksena tähän sääntöön ovat Java Scriptillä suoritettavat, valmiisiin kaavoihin pohjautuvat, vertailut - Esimerksiksi sähköpostiosoitteen muoto, sen oikeellisuutta tai yksilöllisyyttä ei kuitenkaan pystytä tarkistamaan.

AJAX-mallissa sivu kykenee reagoimaan käyttäjän syöttämiin arvoihin jo ennen lomakkeen lähettämistä, tai jopa käyttäjän syöttäessä arvoja. Tämä mahdollistaa

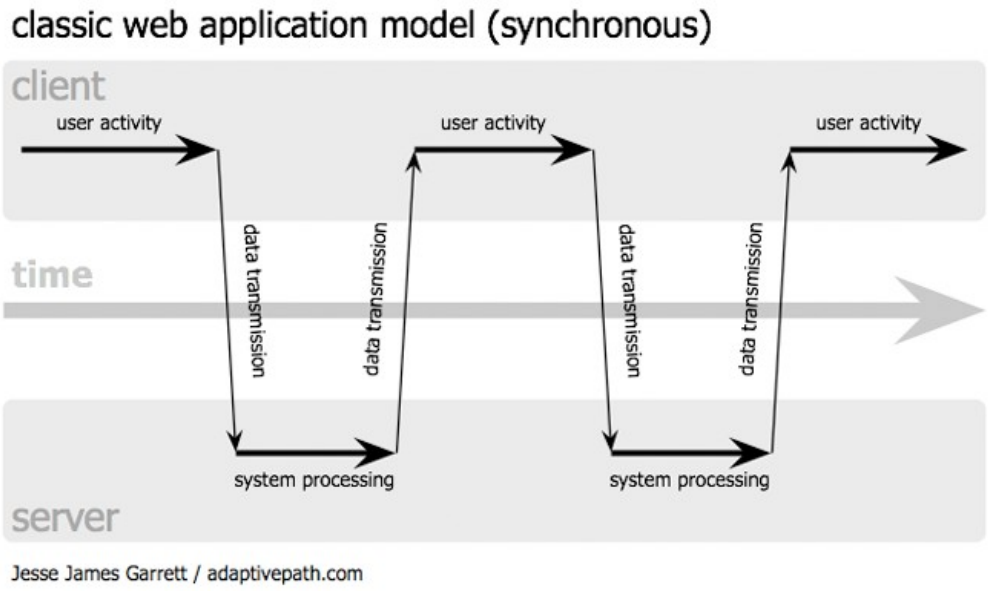
paremman tehokkaamman käyttöliittymän, koska käyttäjän ei tarvitse lähettää lomaketta palvelimelle vaan lomake itse suorittaa syötettyjen arvojen tarkastamisen ja antaa käyttäjälle palautteen välittömästi.

3.2 Vasteajat

Vasteajat käyttäjien näkökulmasta, web-palveluiden kontekstissa, tarkoittavat kahta asiaa: *palvelun vastaamista käyttäjän antamiin toimintoihin* (aika jossa sivusto reagoi napin painallukseen) sekä *toiminnon suoritusaikaa*.

4 Perinteinen web-palvelu

Perinteisessä palvelussa kaikki data ladataan sivupäivityksen yhteydessä; palvelimella sijaitsevat ohjelmapalaset tulkitsevat annetun osoitteen, kokoavat näytettävän ja lähettävät sen käyttäjän selaimeen.



Kuva 2: Jesse James Garretin malli perinteisestä sovellusmallista (Garrett, Jesse James 2005)

4.1 Käyttöliittymä

Perinteisessä rakenteessa sivut ovat täysin staattisia; jos jotain osaa sivusta halutaan muuttaa, on koko sivu ladattava uudelleen.

4.2 Vasteajat

Perinteisessä rakenteessa vasteajat ovat muodostuneet linkin aktivoinnin ja seuraavan sivun valmistumisen välisestä ajasta, mistä johtuen vasteaika voi olla, palvelun rakenteesta riippuen, hyvinkin pitkä (jopa 5 - 10 sekuntia). Tämä aika koostuu web-palvelun yhden kokonaisen sivun lataamisesta, jolloin web-palveluissa on perinteisesti pyritty välttämään suuria määriä kuvia. Tästä johtuen rikkaiden internetsovellusten tuottaminen on ollut aikaisemmin mahdotonta, koska jokaisen tapahtuman yhteydessä

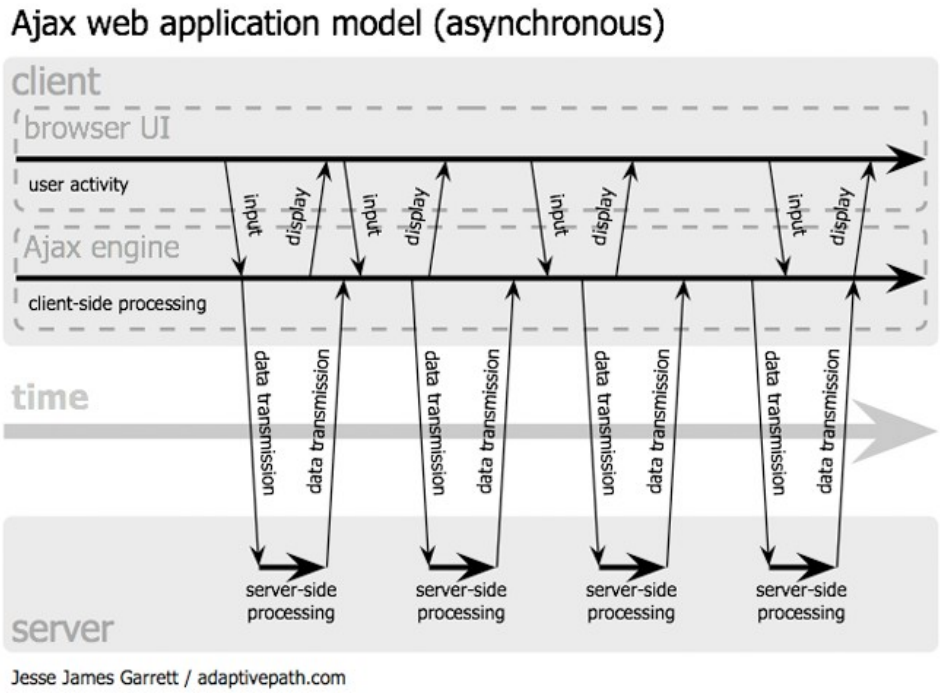
koko käyttöliittymä on ladattava uudestaan, jolloin sovelluksen käyttäminen muuttuu hitaaksi ja epämieluisaksi.

4.3 Esimerkki – Pudotusvalikot

Sivulle, jossa on monta toisistaan riippuvaa pudotusvalikkoa (esimerkiksi maanosa, maa, kaupunki) on klassisen rakenteen mukaisesti kaksi toteutusvaihtoehtoa, *joko ladataan valikoiden kaikki yhdistelmät sivun mukana* tai *päivitetään koko sivu (ja seuraava pudotusvalikko) kun käyttäjä on valinnut pudotusvalikosta haluamansa arvon*. Ensimmäisen mallin mukaisesti valikoiden kaikki yhdistelmät on ladattava sivun mukana. (Kaikki maanosat ja niiden maat sekä kaupungit)

5 Interaktiivinen web-palvelu

Interaktiivinen palvelun pohjalla on yleensä perinteinen rakenne, jossa palvelin rakentaa sivun käyttäjän antaman osoitteen mukaisesti. Interaktiivinen palvelu pystyy kuitenkin myös vaihtamaan halutessaan dataa palvelimen kanssa ilman sivupäivitystä.



Kuva 3: Jesse James Garretin kaavio asynkronisesta sovellusmallista (Garrett, Jesse James 2005)

5.1 Käyttöliittymä

AJAX-rakenteeseen perustuvassa palvelussa käyttöliittymästä voidaan rakentaa huomattavasti tehokkaampi, jopa niinkin paljon että internetsovellukset pystyvät kilpailemaan työpöytäsovellusten kanssa.

5.2 Vasteajat

AJAX-rakenne taas mahdollistaa pelkän halutun datan noutamisen, jolloin vasteajat voidaan leikata jopa sadasosaan perinteisen mallin vasteajasta, koska käyttäjän ei tarvitse odottaa koko sivun lataamista vaan palvelu päivittää vain käyttäjän toimien

vaatiman osan. Tämä on tärkeää, koska käyttöliittymäsuunnittelun mukaisesti käyttäjä kokee välittömänä toiminnon, joka tapahtuu 10 millisekunnin sisällä sen aktivoinnista. Perinteisellä mallilla tällaiset vasteajat ovat mahdottomia, ja käyttäjän saama kokemus tuotteesta heikkenee.

5.3 Esimerkki – Pudotusvalikot

AJAX mahdollistaa pudotusvalikoiden sisällön päivittämisen ”lennossa”, ilman sivun lataamista uudelleen, näin seuraavan valikon vaihtoehdot voidaan ladata käyttäjän tehtyä valintansa edellisessä valikossa. Näin toimiva web-sivu latautuu nopeammin ja tarjoaa silti saman ”drill-down” ominaisuuden.

6 Tutkimus

Ennen varsinaista demon toteutusta tein tutkimusta muista sivuista: niiden käyttöliittymistä sekä toimintaperiaatteista. Keskityin erityisesti sosiaalisiin uutispalveluihin sekä AJAX toteutuksiin. Tarkoituksenani oli tutkia mitä eri tekniikoita sivustot käyttävät ja kuinka tiettyjä ohjelmistoteknisiä ongelmia oli lähestytty.

6.1 Työkalut

Pääasiallisena analysoinnin työkaluina käytin black-box -metodia – Tutkimalla miten sivusto reagoi sille annettuun syötteeseen. Lisäksi käytin *Firefox* -selaimen *Firebug* lisäosaa, joka mahdollistaa muunmuassa palvelimen ja selaimen välillä tapahtuvan liikenteen yksityiskohtaisen tarkastelun.

6.2 Digg.com

Yksi suurimmista tutkimuksen kohteista oli Digg.com; tämän hetken suurin sosiaalinen uutispalvelu, siis suora kilpailija Cloud -palvelulle. Koska Digg ei juurikaan käytä AJAX tekniikoita käyttöliittymässään, keskittyi tutkimukseni lähinnä teknisen toteutuksen black-box tutkimukseen – Syötin sivustolle erilaisia web-osoitteita ja tutkin kuinka palvelu vastaa niihin.

Uuden linkin lisääminen tapahtuu Digg -palvelussa neljässä selkeässä vaiheessa

1. **Osoitteen lähettäminen**
   Käyttäjä lisää osoitteen käyttöliittymän tarjoamaan kenttään
2. **Osoitteen ja sen sisällön tutkiminen**
   Sivusto tarkistaa jos annettu osoite on jo kertaalleen lisätty – Samaa osoitetta pysty lisäämään useaan otteeseen. Mikäli näin ei ole, sivusto hakee osoitteen sisällön ja pyrkii etsimään sieltä otsikon sekä sisällön kuvauksen.
3. **Osoitteen otsikon ja kuvauksen syöttö tai sen vahvistaminen**
   Mikäli otsikkoa ja/tai sisällön kuvausta ei kyetty päättelemään, joutuu käyttäjä syöttämään ne itse. Otsikon ja/tai osoitteen löytyessä annetaan käyttäjälle mahdollisuus halutessaan muokata niitä.
4. **Osoitteen tallentaminen**
   Käyttäjän lähetettyä osoitteen otsikko sekä kuvaus, ne tallennetaan sivuston tietokantaan.

Päätin käyttää kyseistä tekniikkaa Cloud -palvelussa, koska sillä pystyin esittelemään asynkronisen lataamisen hyödyntämistä askeleittain etenevässä prosessissa.

6.3 Gmail

Googlen Gmail palvelu on tunnettu sen vahvasta AJAX-tekniikoiden käytöstään, niinpä oli luonnollista että tutkimukseni sivusi myös kyseistä palvelua.

Gmail.com sivusto itsellään on lähes tyhjä sivu, koko käyttöliittymä ladataan sivulle dynaamisesti AJAX-tekniikoita hyväksi käyttäen, niinpä sen analysoiminen varsin monimutkaista. Analyysin yhteydessä nousi kuitenkin yksi selkeä teema esille – Hidastettu lataaminen. Käyttäjän avatessa Gmail.com sivusto, sovellus näyttää ensin käyttäjälle perus käyttöliittymän ja alkaa tämän jälkeen lataamaan taustalla eri toimintojen vaatimia osia, kuten esimerkiksi tiedoston lataus käyttöliittymä. Lisäksi Google Analytics käyttää samaa metodia – Analytics koodi ladataan Googlen palvelimelta vasta hetki sivuston lataamisen jälkeen.

7 Ratkaisuja käyttäjäkokemuksen parantamiseen

Tässä luvussa esittelen tutkimuksen pohjalta nousseita ratkaisuja käyttäjäkokemuksen parantamiseen.

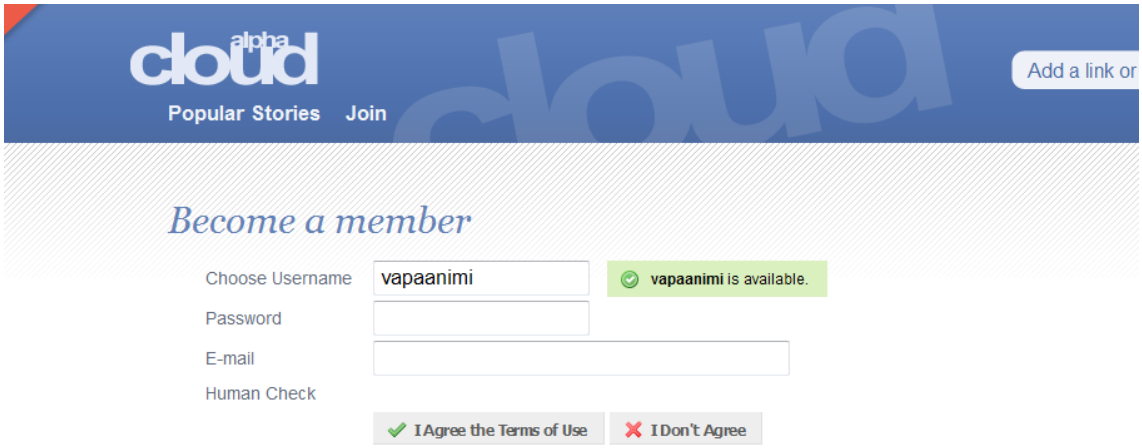
7.1 Asynkroninen sisällön lataaminen

Asynkronisella lataamisella tarkoitetaan tilannetta, jossa dataa ladataan palvelimelta taustalla, ilman sivupäivitystä, kuten perinteinen rakenne vaatisi.

7.2 Rikkaat käyttöliittymät

Rikkailla käyttöliittymillä tarkoitetaan yleisesti sovelluksia, joiden käyttöliittymä käyttäytyy kuten työpöytäsovelluksen käyttöliittymältä voisi odottaa, ja ne ovat toteutettu web-standardeja noudattaen. Perinteisen rakenteen mukaisilla web-sovelluksilla tällaiset käyttöliittymät eivät olleet mahdollisia, joten niitä alettiin kutsumaan rikkaiksi (sovelluksiksi).

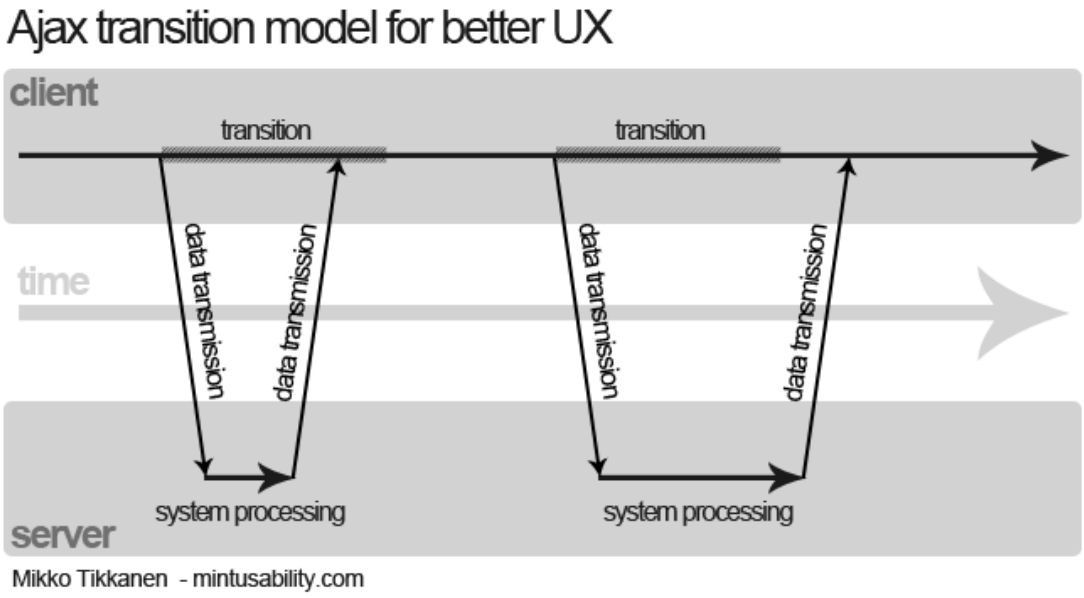
Rikkailla käyttöliittymillä voidaan tehostaa sovelluksen toimintaa ja poistaa muunmuassa tarpeettomia sivulatauksia, esimerkiksi toteuttamalla interaktiivisia lomakkeita, jotka reagoivat välittömästi käyttäjän syöttämiin arvoihin. Tällainen toiminnallisuus ennaltaehkäisee virhetilanteita ja antaa käyttäjälle mahdollisuuden reagoida palautteeseen välittömästi eikä vasta lomakkeen lähettämisen jälkeen, kuten perinteisellä mallilla toteutetuissa sovelluksissa.



Kuva 4: Vuorovaikutteinen lomake, joka on juuri tarkastanut käyttäjänimen yksilöllisyyden

7.3 Transitiot

Transitioilla tarkoitetaan animoituja siirtymiä sisällöstä toiseen. Transitiona voidaan käyttää esimerkiksi liu'utusta, jossa vanha sisältö liu'utetaan näkymättömiin ja uusi sisältö tilalle.



Kuva 5: AJAX transitiomalli

Transitiomalli kuvaa käyttäjän havainnoiman vasteen (mitattu vasteaika - transition kesto) suhdetta mitattuun vasteeseen (vasteaika). Mallin ensimmäinen tapaus

havainnollistaa tilanteen, jolloin verkkoviive sekä prosessointiaika jäävät alle transition ajan, jolloin käyttäjän havaitsema vasteaika on 0 (ts. transition loppuessa, toiminto on suoritettu). Toinen tapaus taas kuvastaa tilannetta, jossa toiminto vaatii palvelimelta enemmän suoritusaikaa. Tällöin verkkoviive sekä prosessointiaika on pidempi kuin transitio, jolloin käyttäjän havaitsee vasteajan. Malli kuitenkin osoittaa, kuinka transitio lyhentää käyttäjän kokemaa vasteaikaa transition keston verran (käyttäjän kokema vasteaika = mitattu vasteaika - transition kesto), jolloin käyttäjän havainnoima vasteaika jää todelista vasteaikaa lyhyemmäksi.

7.4 Hidastettu lataaminen

AJAX:lla voidaan myös toteuttaa ns. hidastettu lataaminen. Tällä tarkoitetaan tekniikkaa, jossa vain kriittisin sisältö ladataan käyttäjän lähettämän pyynnön yhteydessä (esim. sähköpostin sisältö) ja loppu sisältö/metadata myöhemmin toisella kutsulla (esim. muut tiedot). Näin pyritään lyhentämään käyttäjän kokemaa vasteaikaa.

Tekniikkaa käytettäessä on oltava tarkasti selvillä, mitä sisältöä käyttäjä odottaa näkevänsä välittömästi ja mikä voidaan laskea toissijaiseksi sisällöksi. Sillä väärin toteutettuna tekniikalla voidaan saada myös päinvastainen vaikutus, jossa käyttäjä kokee sisällön latautuvan hitaasti.

Lisäksi hidastettua lataamista voidaan hyödyntää myös lisäsisällön esilataamisessa: Esimerkiksi galleria sovelluksessa voitaisiin seuraava kuva ladata valmiiksi taustalla, jolloin käyttäjän siirtyessä seuraavaan kuvaan, ei latausviivettä ole lainkaan. Tekniikkaa täytyy kuitenkin käyttää varoen, vain sellaisissa tilanteissa joissa käyttäjä suurella todennäköisyydellä pyytää lisäsisältöä, muuten luodaan vain ylimääräistä verkkoliikennettä palvelimelle, joka taas nostaa palvelin- sekä yhteyskuluja.

8 Mediateko - Cloud

8.1 Yleistä

Kuva 6: Cloud -projektin etusivu

Cloud työnimellä kulkeva projekti on demo sosiaalisesta uutispalvelusta, johon käyttäjät voivat lähettää linkkejä mielestään kiinnostavista ja/tai tärkeistä uutisista. Kun uutisia alkaa kertyä, ne järjestetään suosion mukaan (kuinka moni on joko linkin lisännyt tai osoittanut pitävänsä sen sisällöstä), jolloin palvelun tarjoama uutisvirta on täysin käyttäjien luomaa, ajantasaista 24 tuntia vuorokaudessa, reagoiden tapahtumiin ympäri maailmaa.

8.2 Toteutustapojen tutkiminen

Päätin myös tutkia mahdollisuutta käyttää valmista PHP rajapintaa demoa varten. Käytyäni läpi muutamia eri vaihtoehtoja (*Code Igniter, Zend, Solar*), potentiaalisimmaksi ehdokkaaksi nousi kiutenkin *Code Igniter*, sen kevyen luonteen ja vähäisen overheadin takia.

Syvennyttyani *Code Igniter*:n dokumentaatioon, se osoittautui kuitenkin ohjelmointiteknisesti hieman kömpelöksi (pääosin super-objekti -toteutuksen johdosta) niinpä päätin rakentaa tätä projektia varten oman rajapinnan.

8.3 Toteutus

Tutkimuksen pohjalta päätin rakentaa projektia varten, PHP päälle oman rajapinnan, pohjautuen osittan *Code Igniter* -rajapintaan. Jotta tehtävä ei karkaisi käsistä, vaan pysyisi riittävän pienenä projektiin nähden, päätin pitää rajapinnan mahdolisimman yksinkertaisena – Yksinkertainen sivupohja järjestemä MVC-mallia (Model View Controller) sivuten sekä sisällön parsiminen annetun URL-osoitteen perusteella. Näin pystyin tarjoilemaan saman sisällön helposti sekä normaalisti että asynkronisesti – yläpaneeliin ladattuna. Ohjelmistoalusta sisältää myös syötteen käsittelyn (GET ja POST), PDO:lla (PHP Data Object) toteutetun MySQL tietokantarajapinnan sekä tietokantaan pohjautuvan sessioiden ja käyttäjien hallinnan. Lisäksi rajapinta tarkastelee millä keinoin sen tarjoamaa sisältöä on pyydetty; Asynkronisen latauksen tapauksessa tarjoillaan pelkkä sisältö, ilman sivupohjaa.

AJAX rajapinnan vaativuuden takia en halunnut toteuttaa sitä itse, vaan päätin käyttää valmista tuotetta. Valitsin *MooTools* -nimisen frameworkin koska se käyttää olio-ohjelmointi tekniikkaa, ollen siten samankaltainen rakentamani PHP-rajapinnan kanssa. Lisäksi, sen olio rakenne mahdollistaa, periytys tekniikkaa käyttäen, juuri halutunlaiset toteutukset.

8.3.1 Työkalut

Ulkoasun ja käyttöliittymän toteutuksessa sekä koostamisessa käytin *Adobe Photoshop* sekä *Eclipse* ohjelmistoja. PHP toteutuksen tein *Eclipse* -sovelluksella, lisäksi käytin *PHPQuickProfiler* -sovellusta sivuston nopeaan profilointiin sekä *FirePHP* -sovellusta debug toiminnoissa (*FirePHP* mahdollistaa PHP tulostuksen suoraan *Firebug*:n konsoliin). Palvelimella käytin Apache, PHP sekä MySQL -sovelluksia.

8.3.2 Ulkoasu ja käyttöliittymä

Ulkoasun luonnostelin ensin paperille, joka jälkeen pilkoin sen pienempiin palasiin, koostamista varten. Jotta projektissa voitaisiin demonstroida tutkimuksen tuottamia tekniikoita päätin sijoittaa sivuston ylälaitaan sivun yläpuolella sijaitsevan paneelin, joka olisi oletuksena piilossa näytön yläreunan yläpuolella. Tämä mahdollistaa sisällön lataamisen asynkronisesti kyseiseen paneeliin, sen ollessa vielä piilossa, näin käyttäjä ei näe itse lataamisprosessia. Päätin myös animoida tämän paneelin, jolloin havaittuja latausaikoja saataisiin lyhennettyä, transitiomallin mukaisesti.

Kuva 7: Yläpaneeli avattuna

8.3.3 Sisällön toteutus

Kun taustatyö oli tehty, lähdin toteuttamaan varsinaista sisältöä sivustolle: uutislistaa, uuden uutisen lisäämistä sekä sisäänkirjautumista – Sivuja joissa voisin hyödyntää tutkimusvaiheessa ilmenneitä tekniikoita. Teknisesti toteutus tapahtui PHP:llä, aikaisemmin rakentamani rajapinnan päälle.

Sisältösivut toteuttavat kaksi eri metodia: normaalia ja asynkronista lataamista varten jotta sisältösivun olisi mahdollista reagoida eri tavoin riippuen siitä miten se on ladattu, näin toteutettuna esimerkiksi AJAX-taso saadaan toimimaan luontevasti.

8.3.4 AJAX toteutus

AJAX toteutuksessa päätin käyttää ”kauniisti hajoavaa” (gracefully degradable) tekniikkaa – Jos käyttäjä saapuu palveluun selaimella joka ei tue AJAX-tekniikoita, voi hän silti käyttää palvelua normaalin web-sivuston tapaan. Kehityksen kannalta tämä tarkoittaa tilannetta jossa sivusto kehitetään ensin toimimaan ilman AJAX-tekniikoita, normaaliin tapaan sivulta sivulle siirtymällä tai sivuja uudelleen lataamalla. Kun palvelu toimii näiltä osin, lisätään mukaan AJAX-taso, joka muuttaa sivuston toiminnallisuuden ”lennossa” AJAX-mallin mukaiseksi. Näin toteutettuna, käyttäjälle, jolla AJAX-taso ei toimi, sivusto toimii edelleen normaalisti, muiden pystyessä silti hyödyntämään AJAX-toteutuksia.

Yläpaneelin liuku transition toteutin käyttäen *MooTools* -rajapinnan tarjoamia metodeja.

Yhteenveto

Monet nykypäivän web-sovelluksista kärsivät vielä suurista ongelmista käyttäjäkokemuksen suhteen, aina hitaista latausajoista tehottomiin käyttöliittymiin, kaikki nämä ongelmat ovat kuitenkin mahdollista kiertää Ajax-tekniikoita käyttäen. Vanhanaikainen jakelutapa (käyttäjä lataa/ostaa sovelluksen, asentaa sen, mahdollisesti asentaa päivitykset jne.) johtaa erittäin fragmentoituneeseen sovelluskantaan sekä monimutkaiseen jakelumalliin, onkin lähes väistämätöntä että tulevaisuudessa suurin osa työpöytäsovelluksista tulee siirtymään verkkoon, web-sovelluksiksi. Varsinkin kun selainten kehittyessä on mahdollista tarjota lähes työpöytämäinen käyttäjäkokemus, nopeine vasteaikoineen ja rikkaine käyttöliittymineen. Lisäksi web-sovellukset mahdollistavat sovellusten mukauttamisen eri mobiilialustoille, jolloin Ajax-sovellusten suuremmat tuotantokustannukset ovat perusteltuja.

Lähteet

Wikipedia. 2009. Ajax (programming) [15.11.2009]
http://en.wikipedia.org/wiki/Ajax_(programming)

Garrett, Jesse James 2003. The Elements of User Experience

Garrett, Jesse James. 2005. Ajax: A New Approach to Web Applications [17.1.2010]
http://adaptivepath.com/ideas/essays/archives/000385.php